# Лекція 2. ОСНОВНІ СТАНДАРТНІ МОДУЛІ PYTHON

* 1. **Поняття модуля.**

Якщо залишити інтерпретатор Python, а потім увійти до нього знову, усі зроблені визначення функцій та змінних буде втрачено. Тому, якщо потрібно написати дещо довшу програму, то для цього краще використовувати текстовий редактор в якому виконану програму потрібно записати у файл. Це називається створенням скрипта. Коли програма збільшується, для полегшення її підтримки слід розбити її на кілька файлів.

Для підтримки цього, Python має певний механізм для створення визначень у файлі, які згодом можуть використовуватися у скрипті чи у діалоговому режимі інтерпретатора. Такий файл зветься *модулем*. Визначення, задані в модулі, імпортуються в інші модулі або в основний (*main*) модуль, який являє собою сукупність об'єктів.

*Модуль* – це файл, що складається з описів функцій та інструкцій Python. Назва файла є назвою модуля, до якої додається розширення .py. Всередині модуля його назва доступна через значення глобальної змінної *\_\_name\_\_*. Для прикладу, створимо у текстовому редакторі файл *fibo.py* у поточній директорії з таким змістом:

***# Модуль, що обчислює числа Фібоначчі***

***def fib(n): # виводить числа Фібоначчі до n***

***a, b = 0, 1***

***while b < n:***

***print(b)***

***a, b = b, a+b***

***def fib2(n): # повертає числа Фібоначчі до n***

***result = []***

***a, b = 0, 1***

***while b < n:***

***result.append(b)***

***a, b = b, a+b***

***return result***

Тепер відкриємо інтерпретатор Python та імпортуємо цей модуль за допомогою такої команди:

***>>> import fibo***

Ця команда додає в поточний простір імен лише саму назву модуля *fibo*, а не назви функцій, визначених у ньому. Використовуючи назву модуля ви маєте доступ до його функцій:

***>>> fibo.fib(1000)***

***1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987***

***>>> fibo.fib2(100)***

***[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]***

***>>> fibo.\_\_name\_\_***

***'fibo'***

Якщо часто використовувати функцію, то їй краще дати локальну назву:

***>>> fib = fibo.fib***

***>>> fib(500)***

***1 1 2 3 5 8 13 21 34 55 89 144 233 377***

* 1. **Модулі в python.**

Модуль може містити не лише описи функцій, а й виконувані інструкції. Ці інструкції потрібні для ініціалізації модуля. Вони виконуються лише при першому імпортуванні модуля. (Насправді визначення функцій – це також інструкції, що "виконуються"; виконання ж полягає в тому, що назва функції вводиться у глобальний простір імен модуля).

Кожен модуль має свій власний простір імен, який використовується як глобальний усіма визначеними у цьому модулі функціями. Таким чином, автор модуля може використовувати глобальні змінні всередині модуля, уникнувши при цьому можливого конфлікту з глобальними змінними, що задані користувачем модуля. З іншого боку, ви можете дістатися до глобальних змінних модуля за допомогою тієї ж нотації, що використовується для доступу функцій:

***назва\_модуля.елемент\_модуля***

Модулі можуть імпортувати інші модулі. Зазвичай, хоча це і не є необхідним, всі інструкції *import* пишуть на початку модуля. Імпортовані назви модулів додаються до глобального простору імен модуля.

Існує варіант інструкції *import*, який напряму імпортує назви з модуля у простір імен імпортуючого модуля. Наприклад:

***>>> from fibo import fib, fib2***

***>>> fib(500)***

***1 1 2 3 5 8 13 21 34 55 89 144 233 377***

Ця операція не додає до локальної символьної таблиці назву самого модуля, з якого відбувся імпорт (зокрема, у цьому прикладі назва fibo – невизначена).

Існує також можливість для імпорту всіх назв, визначених у модулі:

***>>> from fibo import \****

***>>> fib(500)***

***1 1 2 3 5 8 13 21 34 55 89 144 233 377***

При цьому імпортуються усі назви, крім тих, що починаються з символу підкреслювання (\_).

* + 1. Шлях пошуку модулів

Якщо імпортується модуль, який називається *spam*, то інтерпретатор спочатку шукає файл з назвою *spam.py* у поточній директорії, а потім у директоріях, визначених змінною середовища *PYTHONPATH*. Вона має такий же синтаксис, як і змінна оболонки *PATH*, тобто являє собою список директорій. Якщо *PYTHONPATH* не задано, або якщо файл там не знайдено, то пошук продувжується за типовою адресою, яка залежить від інсталяції; у системах *Unix* це здебільшого */usr/local/lib/python*.

Тут слід уточнити, що пошук модулів починається зі списку директорій, заданих змінною *sys.path*, яка ініціалізується директорією, де розміщено скрипт вводу (чи у поточній директорії), а потім доповнюється з *PYTHONPATH* та залежним від інсталяції шляхом. Це дозволяє програмам змінювати адресу пошуку. Зауважимо, що оскільки назва директорії, де знаходиться виконуваний скрипт, є у змінній *sys.path*, тому важливо, щоб назва скрипта не збігалася з назвою певного стандартного модуля, бо інакше Python спробує завантажити скрипт замість модуля при імпорті. Загалом це повинно призвести до помилки.

* + 1. Компільовані файли

Існує можливість значного прискорення запуску коротких програм, що використовують багато стандартних модулів: якщо в директорії, де знаходиться файл *spam.py*, існує файл *spam.pyc*, то вважається, що він містить скомпільовану в байт-код версію модуля *spam*. Час останьої зміни версії *spam.py*, що використовується для створення *spam.pyc*, записується у *spam.pyc*, і файл *.pyc* пропускається, якщо час зміни скомпільованої версії не відповідає текстовій.

У більшості випадків для створення файла *spam.pyc* взагалі не потрібно нічого робити. Як тільки *spam.py* скомпільовано, інтерпретатор зробить спробу записати скомпільовану версію у *spam.pyc*. Якщо ця спроба не вдається, то це не призводить до помилки. Якщо з певних причин файл не записано повністю, то новостворений *spam.pyc* буде вважатися недійсним і таким чином не буде використовуватися пізніше. Вміст файла *spam.pyc* не залежить від платформи, отже директорія, що містить модулі, написані на Python, може використовуватися машинами різних архітектур.

* + 1. Стандартні модулі

Python має бібліотеку стандартних модулів, яка описана в окремому документі, що зветься "Довідник бібліотеки мови Python" (Python Library Reference) (надалі "Довідник бібліотеки"). Окремі модулі, що вбудовано в інтерпретатор, надають доступ до операцій, що не є основною частиною мови, але все таки вони вбудовані, або для ефективності, або для того, щоб надати доступ до функцій операційної системи, зокрема системних викликів. Такі модулі є частиною конфігурації, яка залежить від платформи.

Один окремий модуль заслуговує спеціальної уваги: *sys*, який вбудовано у будь-який інтерпретатор мови Python.

Змінна *sys.path* – це список рядків, що визначають використовувані інтерпретатором шляхи пошуку модулів. Вона ініціалізується стандартним значенням, яке можна дістати із змінної середовища *PYTHONPATH* або із вбудованого стандартного значення (якщо *PYTHONPATH* не задано). ЇЇ можна змінити за допомогою стандартних операцій зі списками:

***>>> import sys***

***>>> sys.path.append('/ufs/guido/lib/python')***

Вбудована функція *dir()* використовується для виявлення усіх назв, визначених у модулі. Вона повертає впорядкований список рядків:

***>>> import fibo, sys***

***>>> dir(fibo)***

Без аргументів *dir()* повертає назви, визначені на момент виклику функції:

***>>> a = [1, 2, 3, 4, 5]***

***>>> import fibo, sys***

***>>> fib = fibo.fib***

***>>> dir()***

***['\_\_name\_\_', 'a', 'fib', 'fibo', 'sys']***

Зауважимо, що цей список містить всі види ідентифікаторів: змінні, модулі, функції тощо.

*dir()* не видає назв вбудованих функцій та змінних. Якщо ці назви потрібні, то вони визначені у стандартному модулі *\_\_builtin\_\_*:

***>>> import \_\_builtin\_\_***

***>>> dir(\_\_builtin\_\_)***

* 1. **Огляд стандартної бібліотеки.**
     1. Інтерфейс операційної системи

Модуль *os* містить функції взаємодії з операційною системою:

***>>> import os***

***>>> os.system('time 0:02')***

***0***

***>>> os.getcwd() # Повертає поточну робочу директорію***

***'C:\\Python24'***

***>>> os.chdir('/server/accesslogs')***

Пам'ятайте, що слід використовувати "*import os*" замість "*from os import \*".* Це дозволить запобігти перекриванню вбудовано функції *open()* функцією *os.open(),* яка має зовсім інше призначення.

Вбудовані функції *dir()* та *help()* є дуже корисними для отримання допомоги при роботі з такими великими модулями як *os*:

***>>> import os***

***>>> dir(os) # повертає список усіх функцій модуля***

***>>> help(os) # повертає інструкцію створену збиранням до купи рядків документації модуля***

Для щоденних потреб, пов'язаних з файлами та директоріями, модуль *shutil* надає інтерфейс більш високого рівня, що спрощує програмування:

***>>> import shutil***

***>>> shutil.copyfile('data.db', 'archive.db') # копіювання***

***>>> shutil.move('/build/executables', 'installdir') # переміщення***

* + 1. Шаблони розширення файлових назв

Модуль *glob* містить функцію, що дозволяє створювати списки файлів за допомогою шаблонів розширення, застосованих до директорій:

***>>> import glob***

***>>> glob.glob('\*.py')***

***['primes.py', 'random.py', 'quote.py']***

* + 1. Аргументи командного рядка

Скрипти часто використовують аргументи, подані з командного рядка. Ці аргументи зберігаються у вигляді списку атрибута *argv*, що знаходиться в модулі *sys*. Наприклад, якщо з командного рядка було запущено команду "*python demo.py one two three*", то ми можемо отримати такий вивід:

***>>> import sys***

***>>> print sys.argv***

***['demo.py', 'one', 'two', 'three']***

Модуль *getopt* оброблює *sys.argv* на основі конвенцій юніксової функції *getopt()*. Потужнішу і гнучкішу обробку командного рядка можна знайти у модулі *optparse*.

* + 1. Переспрямування виводу помилок та вихід із програми

Модуль *sys* має також атрибути *stdin*, *stdout* та *stderr* ("стандартний ввід", "стандартний вивід" та "стандартний вивід помилок" відповідно ). Останній корисний для виводу попереджень і помилок при переспрямуванні *stdout*:

***>>> sys.stderr.write('Попередження: файл для запису не знайдено; створюється новий файл')***

***Попередження: файл для запису не знайдено; створюється новий файл***

Найпростіший шлях виходу з програми — це виклик "*sys.exit()*".

* + 1. Пошук за шаблоном

Модуль *re* містить утиліти регулярних виразів для пошуку за шаблоном всередині рядків. Регулярні вирази надають компактні оптимальні вирішення при застосуванні доволі складних правил пошуку:

***>>> import re***

***>>> re.findall(r'\bf[a-z]\*', 'which foot or hand fell fastest')***

***['foot', 'fell', 'fastest']***

***>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')***

***'cat in the hat'***

Якщо потрібні лише прості маніпуляції, то найкраще застосовувати методи рядків, які набагато простіше читати:

***>>> 'tea for too'.replace('too', 'two')***

***'tea for two'***

* + 1. Математика

Модуль *math* надає можливість доступу до функцій бібліотеки C для роботи з дробовими числами:

***>>> import math***

***>>> math.cos(math.pi / 4.0)***

***0.70710678118654757***

***>>> math.log(1024, 2)***

***10.0***

Модуль *random* містить утиліти для роботи з випадковими числами:

***>>> import random***

***>>> print random.choice(['яблуко', 'груша', 'банан'])***

***'яблуко'***

***>>> random.sample(xrange(100), 10) # вибір без заміщення***

***[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]***

***>>> random.random() # випадкове число з рухомою комою***

***0.17970987693706186***

***>>> random.randrange(6) # випадкове ціле число, вибране з послідовності range(6)***

***4***

* + 1. Доступ до мережі Інтернет

Існують кілька модулів для доступу до інтернету та обробки його протоколів. Два найпростіші — це *urllib2* (для отримання даних з інтернет-адрес) та *smtplib* для відправлення електронної пошти:

***>>> import urllib2***

***>>> for line in urllib2.urlopen('http://tycho.usno.navy.mil/cgi-bin/timer.pl'):***

***... if 'EST' in line: # шукаємо Eastern Standard Time***

***... print line***

***Nov. 25, 09:43:32 PM EST***

***>>> import smtplib***

***>>> server = smtplib.SMTP('localhost')***

***>>> server.sendmail('soothsayer@tmp.org', 'jceasar@tmp.org',***

***"""To: jceasar@tmp.org***

***From: soothsayer@tmp.org***

***Beware the Ides of March.***

***""")***

***>>> server.quit()***

* + 1. Час і число

Модуль *datetime* містить класи для роботи з даними, що виражають час та число, як у складний так і в простий спосіб. Він придатний і для арифметики часових даних, хоча основна увага приділяється тому, щоб ефективно дістати дані для форматування та їхньої обробки. Модуль також має об'єкти, що розрізняють різні часові зони.

***# створення та форматування чисел дуже просте***

***>>> from datetime import date***

***>>> now = date.today()***

***>>> now***

***datetime.date(2003, 12, 2)***

***>>> now.strftime("%m-%d-%y or %d%b %Y is a %A on the %d day of %B")***

***'12-02-03 or 02Dec 2003 is a Tuesday on the 02 day of December'***

***# часові дані придатні для застосування календарної арифметики***

***>>> birthday = date(1964, 7, 31)***

***>>> age = now — birthday***

***>>> age.days***

***14368***

* + 1. Ущільнення даних

Поширені формати ущільнення та архівації даних напряму підтримуються такими модулями як *zlib*, *gzip*, *bz2*, *zipfile* та *tarfile*.

***>>> import zlib***

***>>> s = 'witch which has which witches wrist watch'***

***>>> len(s)***

***41***

***>>> t = zlib.compress(s)***

***>>> len(t)***

***37***

***>>> zlib.decompress(t)***

***'witch which has which witches wrist watch'***

***>>> zlib.crc32(t)***

***-1438085031***

2.3.10. Обчислення продуктивності

Окремі користувачі мови Python зацікавлені у тому, наскільки продуктивними є різні підходи вирішення однієї проблеми відносно один одного. До складу Python входить модуль *timeit*, який дозволяє швидко віднайти відповіді на ці питання.

***>>> from timeit import Timer***

***>>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()***

***0.60864915603680925***

***>>> Timer('a,b = b,a', 'a=1; b=2').timeit()***

***0.8625194857439773***

На відміну від високого рівня детальності модуля *timeit*, модулі *profile* та *pstats* мають інструменти для ідентифікації критичних ділянок коду у більших блоках коду.

2.3.11. Контроль якості

Один із способів для створення якісного програмного забезпечення - це написання спеціальних тестів для кожної функції, і часте використання цих тестів під час процесу розробки.

Модуль *doctest* має спеціальні інструменти для сканування модуля та перевірки тестів, що вказані в рядках документації. Створення ж тестів – дуже просте і полягає у копіюванні та вставці типового виклику функції та її результату в рядок документації. Додання прикладу вдосконалює документацію а також дозволяє модулю *doctest* перевірити, чи відповідає код документації:

***def average(values):***

***"""Виводить середнє арифметичне для даного списку чисел.***

***print average([20, 30, 70])***

***40.0***

***"""***

***return sum(values, 0.0) / len(values)***

***import doctest***

***doctest.testmod() # автоматично перевірити тести***

Модуль *unittest* є дещо складнішим за *doctest*, але натомість дозволяє провести більш ґрунтовне тестування за допомогою правил, що здебільшого задаються в окремому файлі:

***import unittest***

***class TestStatisticalFunctions(unittest.TestCase):***

***def test\_average(self):***

***self.assertEqual(average([20, 30, 70]), 40.0)***

***self.assertEqual(round(average([1, 5, 7]), 1), 4.3)***

***self.assertRaises(ZeroDivisionError, average, [])***

***self.assertRaises(TypeError, average, 20, 30, 70)***

***unittest.main() # Виклик з командного рядка запускає всі тести***

2.3.12. Форматування виводу

Модуль *repr* має версію функції *repr()* для скороченого зображення великих або багаторівневих структур даних:

***>>> import repr***

***>>> repr.repr(set('supercalifragilisticexpialidocious'))***

***"set(['a', 'c', 'd', 'e', 'f', 'g', ...])"***

Модуль *pprint* (pretty printer) надає можливість більш досконалого контролю при виводі об'єктів (як вбудованих, так і заданих користувачем) у вигляді, придатному для зчитування інтерпретатором. Якщо результат довший за один рядок, то ця функція додає пробіли та символи нового рядка, які дозволяють ясніше виразити структуру даних:

***>>> import pprint***

***>>> t = [[[['black', 'cyan'], 'white', ['green', 'red']], [['magenta',***

***... 'yellow'], 'blue']]]***

***...***

***>>> pprint.pprint(t, width=30)***

***[[[['black', 'cyan'],***

***'white',***

***['green', 'red']],***

***[['magenta', 'yellow'],***

***'blue']]]***

Модуль *textwrap* форматує текст для певної ширини екрану:

***>>> import textwrap***

***>>> doc = """Метод wrap() подібний до fill(), але він повертає***

***... список рядків замість одного довгого рядка, розбитого***

***... на рядки."""***

***...***

***>>> print textwrap.fill(doc, width=40)***

***Метод wrap() подібний до fill(),***

***але він повертає список рядків***

***замість одного довгого рядка,***

***розбитого на рядки.***

Модуль *locale* завантажує формати даних, специфічні для певного культурного оточення. Спеціальний атрибут форматуючої функції модуля надає можливість прямого форматування чисел за допомогою групових роздільників:

***>>> import locale***

***>>> locale.setlocale(locale.LC\_ALL, 'uk\_UA.utf8')***

***('uk\_UA', 'utf8')***

***>>> conv = locale.localeconv() # отримати правила переведення***

***>>> x = 1234567.8***

***>>> locale.format("%d", x, grouping=True)***

***'1.234.567'***

***>>> print locale.format("%.\*f%s",***

***... (conv['int\_frac\_digits'], x,***

***... conv['currency\_symbol']), grouping=True)***

***1.234.567,80гр***

2.3.13. Шаблони

Модуль *string* має клас *Template* з досить простим синтаксисом, придатним для редагування користувачами. За його допомогою користувачі можуть змінювати текстові величини програми без внесення змін до її коду.

Формат модуля використовує спеціальні назви-заповнювачі, що утворюються за допомогою символа "*$*" та дійсного ідентифікатора мови Python (буквено-цифрові символи та нижня риска). Для відокремлення назви-заповнювача від наступних буквено-цифрових символів слід використовувати фігурні дужки. "*$$*" задає один символ "*$*":

***>>> from string import Template***

***>>> t = Template('${village} витратили $$10 на $cause.')***

***>>> print t.substitute(village='Васюки', cause='сміттєфонд')***

***Васюки витратили $10 на сміттєфонд.***

Метод *substitute* відкидає *KeyError,* якщо ключове слово не існує в словнику або в ключовому аргументі. Для програм, де дані для заповнення шаблонів можуть бути неповними, краще використовувати метод *safe\_substitute*, що за умови відсутності відповідних даних залишить незаповнені назви без змін.

***>>> t = Template('Повернути $item $owner.')***

***>>> d = dict(item='непроковтнутий шматок')***

***>>> t.substitute(d)***

***Traceback (most recent call last):***

***. . .***

***KeyError: 'owner'***

***>>> print t.safe\_substitute(d)***

***Повернути непроковтнутий шматок $owner.***

Детальніше ознайомитися з усіма стандартними бібліотеками Python можна за посиланням <https://docs.python.org/3/library/index.html> .